# app.py (Streamlit Frontend) - UI THEME UPDATE ONLY

import streamlit as st

import pandas as pd

import requests

import io

import time

import base64

import os

from datetime import datetime

import json

import tempfile

import shutil

# FastAPI backend URL

API\_BASE\_URL = "http://localhost:8000"

# Helper functions for deep config

def validate\_and\_normalize\_headers(columns):

"""Validate and normalize column headers"""

new\_columns = []

for i, col in enumerate(columns):

if col is None or str(col).strip() == "":

new\_col = f"column\_{i+1}"

else:

new\_col = str(col).strip().lower()

new\_columns.append(new\_col)

return new\_columns

# ---------- Minimalist Dark Theme with Exact Colors ----------

st.markdown("""

<style>

:root {

--ev-colors-primary: #282828;

--ev-colors-secondary: #424242;

--ev-colors-tertiary: #4e332a;

--ev-colors-highlight: #e75f33;

--ev-colors-text: #fff;

--ev-colors-secondaryText: grey;

--ev-colors-tertiaryText: #a3a3a3;

--ev-colors-borderColor: #ffffff1f;

--ev-colors-background: #161616;

--ev-colors-success: #d8fc77;

--ev-colors-danger: #dc143c;

}

/\* Main background \*/

.stApp {

background: var(--ev-colors-background);

color: var(--ev-colors-text);

}

/\* Headers \*/

h1, h2, h3, h4, h5, h6 {

color: var(--ev-colors-text) !important;

border-left: 4px solid var(--ev-colors-secondary) !important;

padding-left: 10px !important;

}

/\* Cards \*/

.custom-card {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 8px;

padding: 20px;

margin: 10px 0;

transition: all 0.3s ease;

}

.custom-card:hover {

background: var(--ev-colors-secondary);

}

.card-title {

color: var(--ev-colors-text);

font-size: 1.2em;

font-weight: 600;

margin-bottom: 15px;

}

.card-content {

color: var(--ev-colors-tertiaryText);

font-size: 0.95em;

line-height: 1.5;

}

/\* Buttons - Only primary buttons use highlight color \*/

.stButton > button {

background: var(--ev-colors-secondary) !important;

color: var(--ev-colors-text) !important;

border: 1px solid var(--ev-colors-borderColor) !important;

border-radius: 6px !important;

padding: 8px 16px !important;

font-weight: 500 !important;

transition: all 0.2s ease !important;

}

.stButton > button:hover {

background: var(--ev-colors-tertiary) !important;

border-color: var(--ev-colors-tertiaryText) !important;

}

/\* Primary/Important buttons use highlight color \*/

.primary-button > button {

background: var(--ev-colors-highlight) !important;

color: white !important;

border: none !important;

font-weight: 600 !important;

}

.primary-button > button:hover {

background: #f27024 !important;

transform: translateY(-1px) !important;

}

/\* Process steps \*/

.process-step {

background: var(--ev-colors-primary);

padding: 15px;

border-radius: 6px;

margin: 8px 0;

border-left: 4px solid var(--ev-colors-secondary);

transition: all 0.3s ease;

}

.process-step.running {

border-left-color: var(--ev-colors-highlight);

}

.process-step.completed {

border-left-color: var(--ev-colors-success);

}

.process-step.pending {

border-left-color: var(--ev-colors-secondary);

}

/\* Dataframes \*/

.dataframe {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-text) !important;

}

/\* Input fields \*/

.stTextInput > div > div > input {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

.stTextInput > div > div > input:focus {

border-color: var(--ev-colors-highlight);

}

/\* Select boxes \*/

.stSelectbox > div > div {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

/\* Number inputs \*/

.stNumberInput > div > div > input {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

/\* Checkboxes & Radio buttons \*/

.stCheckbox > label, .stRadio > label {

color: var(--ev-colors-text) !important;

}

/\* Sidebar \*/

.css-1d391kg {

background: var(--ev-colors-primary) !important;

}

/\* Messages \*/

.stSuccess {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-success) !important;

border-left: 4px solid var(--ev-colors-success) !important;

}

.stError {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-danger) !important;

border-left: 4px solid var(--ev-colors-danger) !important;

}

.stWarning {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-highlight) !important;

border-left: 4px solid var(--ev-colors-highlight) !important;

}

.stInfo {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-text) !important;

border-left: 4px solid var(--ev-colors-secondary) !important;

}

/\* Text areas \*/

.stTextArea > div > div > textarea {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

/\* Preview table \*/

.preview-table {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 6px;

padding: 15px;

margin: 10px 0;

}

/\* File upload \*/

.uploadedFile {

background: var(--ev-colors-primary);

border: 2px dashed var(--ev-colors-borderColor);

border-radius: 8px;

padding: 20px;

text-align: center;

margin: 10px 0;

}

/\* Progress bar \*/

.stProgress > div > div > div {

background-color: var(--ev-colors-highlight);

}

/\* Scrollable chunk display \*/

.scrollable-chunk {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 4px;

padding: 10px;

margin: 5px 0;

max-height: 300px;

overflow-y: auto;

font-family: monospace;

font-size: 0.85em;

line-height: 1.4;

white-space: pre-wrap;

word-wrap: break-word;

}

.chunk-header {

background: var(--ev-colors-secondary);

padding: 8px 12px;

border-radius: 4px;

margin-bottom: 8px;

font-weight: bold;

color: var(--ev-colors-text);

}

/\* Scrollbar \*/

.scrollable-chunk::-webkit-scrollbar {

width: 6px;

}

.scrollable-chunk::-webkit-scrollbar-track {

background: var(--ev-colors-primary);

}

.scrollable-chunk::-webkit-scrollbar-thumb {

background: var(--ev-colors-secondary);

border-radius: 3px;

}

.scrollable-chunk::-webkit-scrollbar-thumb:hover {

background: var(--ev-colors-tertiaryText);

}

/\* Minimal highlight usage \*/

.highlight-text {

color: var(--ev-colors-highlight);

font-weight: 600;

}

/\* Section headers \*/

.section-header {

color: var(--ev-colors-text);

border-bottom: 1px solid var(--ev-colors-borderColor);

padding-bottom: 10px;

margin-bottom: 20px;

}

</style>

""", unsafe\_allow\_html=True)

# ---------- SVG Logo Integration ----------

logo\_svg = """<svg id="Layer\_2" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1703.31 535.6"><defs><style>

.cls-1 {

fill: #fff;

}

.cls-2 {

fill: #fbb03b;

}

.cls-3 {

fill: #f27024;

}

</style></defs><g id="Layer\_10"><g><path class="cls-1" d="M125.67,428.34c-39.15,0-70.27-13.09-92.48-38.91C11.17,363.84,0,334.47,0,302.15c0-30.4,9.47-57.88,28.14-81.68,23.77-30.39,56.01-45.8,95.83-45.8s74.1,15.76,98.58,46.85c17.39,21.95,26.36,49.63,26.66,82.28l.05,5.23H41.22c1.5,23.04,9.58,42.3,24.08,57.31,15.74,16.28,34.65,24.2,57.81,24.2,11.12,0,22.08-1.96,32.6-5.83,10.49-3.85,19.51-9.02,26.82-15.36,7.36-6.39,8.83-7.95,14.56-15.39l2.6-4.32c5.42-9.02,16.94-12.25,26.26-7.35h0c9.62,5.06,13.39,16.91,8.46,26.6l-1.53,3c-8.02,11.54-10.34,14.39-21.53,24.68-11.22,10.32-24.02,18.29-38.05,23.68-14.02,5.38-30.04,8.1-47.63,8.1ZM204.47,272.93c-3.65-12.13-8.55-22.08-14.6-29.64-7.06-8.82-16.57-16.06-28.27-21.51-11.75-5.46-24.27-8.23-37.2-8.23-21.29,0-39.83,6.92-55.1,20.58-9.88,8.81-17.76,21.84-23.46,38.8h158.64Z"></path><rect class="cls-1" x="288.28" y="97.26" width="40.15" height="331.08" rx="20.07" ry="20.07"></rect><path class="cls-1" d="M490.58,428.34c-39.15,0-70.27-13.09-92.48-38.91-22.02-25.59-33.19-54.96-33.19-87.28,0-30.4,9.47-57.88,28.14-81.68,23.77-30.39,56.01-45.8,95.83-45.8s74.1,15.76,98.58,46.85c17.39,21.95,26.36,49.63,26.66,82.28l.05,5.23h-208.03c1.5,23.04,9.58,42.3,24.08,57.31,15.74,16.28,34.65,24.2,57.81,24.2,11.12,0,22.08-1.96,32.6-5.83,10.49-3.85,19.51-9.02,26.82-15.36,7.36-6.39,8.83-7.95,14.56-15.39l2.6-4.32c5.42-9.02,16.94-12.25,26.26-7.35h0c9.62,5.06,13.39,16.91,8.46,26.6l-1.53,3c-8.02,11.54-10.34,14.39-21.53,24.68-11.22,10.32-24.02,18.29-38.05,23.68-14.02,5.38-30.04,8.1-47.63,8.1ZM569.37,272.93c-3.65-12.13-8.55-22.08-14.6-29.64-7.06-8.82-16.57-16.06-28.27-21.51-11.75-5.46-24.27-8.23-37.2-8.23-21.29,0-39.83,6.92-55.1,20.58-9.88,8.81-17.76,21.84-23.46,38.8h158.64Z"></path><path class="cls-1" d="M751.92,422.82l-96-208.47c-5.97-12.97,3.5-27.77,17.78-27.77h0c7.64,0,14.59,4.45,17.78,11.39l69.08,150.01,68.21-149.93c3.18-6.99,10.15-11.47,17.82-11.47h.22c14.26,0,23.74,14.76,17.8,27.73l-95.43,208.49c-1.55,3.38-4.92,5.54-8.63,5.54h0c-3.71,0-7.08-2.16-8.63-5.52Z"></path><g><path class="cls-2" d="M1052.79,311.55c-30.67,0-56.25,33.01-62.14,66.95,5.07-11.19,11.63-17.94,18.79-17.94,15.94,0,23.38,33.67,28.84,74.37,1.51,11.28,12.67,86.53,13.56,100.67.05,0,.11,0,.16,0,1.04-16.27,10.83-87.61,12.64-100.66,5.78-41.56,12.93-74.37,28.87-74.37,9.09,0,17.21,10.84,22.5,27.76-2.22-38.69-29.66-76.77-63.22-76.77Z"></path><path class="cls-3" d="M1053.33,46.78c60,50.38,96.73,131.67,97.74,218.86-26.55-32.52-60.86-50.27-97.76-50.27s-71.19,17.74-97.74,50.24c1.01-87.19,37.75-168.47,97.75-218.83M1053.33,0c-80.86,53.76-135.27,154.25-135.27,269.32,0,28.59,3.36,56.29,9.66,82.6,4.47,18.64,10.39,36.6,17.66,53.67,2.54-84.98,49.89-152.72,107.94-152.72s105.41,67.76,107.94,152.76c10.02-23.52,17.51-48.73,22.09-75.13,3.46-19.78,5.25-40.25,5.25-61.19C1188.59,154.25,1134.19,53.78,1053.33,0h0Z"></path></g><path class="cls-3" d="M1246.12,390.85l-15.96-370.06C1229.55,9.49,1238.55,0,1249.87,0h0c11.31,0,20.31,9.49,19.71,20.79l-15.96,370.06h-7.5Z"></path><path class="cls-1" d="M1333.96,408.27v-185.58h-40.62v-36.1h40.62v-69.25c0-11.09,8.99-20.07,20.07-20.07h0c11.09,0,20.07,8.99,20.07,20.07v69.25h62.21v36.1h-62.21v185.58c0,11.09-8.99,20.07-20.07,20.07h0c-11.09,0-20.07-8.99-20.07-20.07Z"></path><path class="cls-1" d="M1579.72,428.34c-39.15,0-70.26-13.09-92.48-38.91-22.02-25.59-33.18-54.95-33.18-87.28,0-30.4,9.47-57.88,28.14-81.68,23.77-30.39,56.01-45.8,95.83-45.8s74.1,15.76,98.59,46.85c17.39,21.94,26.36,49.63,26.66,82.28l.05,5.23h-208.03c1.5,23.04,9.59,42.3,24.08,57.31,15.74,16.28,34.64,24.2,57.81,24.2,11.12,0,22.09-1.96,32.6-5.83,10.49-3.85,19.51-9.02,26.82-15.36,7.36-6.39,9.22-7.53,15.54-17.02l1.62-2.69c5.42-9.02,16.94-12.25,26.26-7.35h0c9.62,5.06,13.39,16.91,8.46,26.60l-1.36,2.67c-6.09,8.44-10.51,14.72-21.7,25.01-11.22,10.32-24.02,18.29-38.06,23.68-14.02,5.38-30.04,8.1-47.63,8.1ZM1658.52,272.93c-3.65-12.13-8.55-22.08-14.6-29.64-7.06-8.82-16.57-16.06-28.27-21.51-11.76-5.46-24.27-8.23-37.2-8.23-21.29,0-39.83,6.92-55.1,20.58-9.89,8.81-17.76,21.85-23.46,38.8h158.64Z"></path></g></g></svg>"""

# Convert SVG to base64 and display

b64\_logo = base64.b64encode(logo\_svg.encode('utf-8')).decode("utf-8")

# Display logo and header

st.markdown(

f'''

<div style="text-align: center; margin-bottom: 20px;">

<img src="data:image/svg+xml;base64,{b64\_logo}" width="300" alt="I Chunk Optimizer Logo">

</div>

<div style="background: var(--ev-colors-primary); border: 1px solid var(--ev-colors-borderColor); border-radius: 8px; padding: 20px; margin-bottom: 30px;">

<h1 style="color: var(--ev-colors-text); text-align: center; margin: 0; font-size: 2.2em;">I Chunk Optimizer</h1>

<p style="color: var(--ev-colors-tertiaryText); text-align: center; margin: 10px 0 0 0; font-size: 1.1em;">Advanced Text Processing + 3GB File Support + Performance Optimized</p>

</div>

''',

unsafe\_allow\_html=True

)

# ---------- ALL REMAINING CODE STAYS EXACTLY THE SAME ----------

# [ALL THE EXISTING API FUNCTIONS AND MAIN APPLICATION CODE REMAINS UNCHANGED]

# [NO FUNCTIONAL CHANGES - ONLY THE CSS THEME WAS UPDATED]

def call\_fast\_api(file\_path: str, filename: str, db\_type: str, db\_config: dict = None,

use\_openai: bool = False, openai\_api\_key: str = None, openai\_base\_url: str = None,

process\_large\_files: bool = True, use\_turbo: bool = False, batch\_size: int = 256):

"""Send CSV upload or trigger DB import for Fast mode"""

try:

# DB import path: send only form data (no file open)

if db\_config and db\_config.get('use\_db'):

data = {

"db\_type": db\_config.get("db\_type"),

"host": db\_config.get("host"),

"port": db\_config.get("port"),

"username": db\_config.get("username"),

"password": db\_config.get("password"),

"database": db\_config.get("database"),

"table\_name": db\_config.get("table\_name"),

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"process\_large\_files": process\_large\_files,

"use\_turbo": use\_turbo,

"batch\_size": batch\_size

}

response = requests.post(f"{API\_BASE\_URL}/run\_fast", data=data)

return response.json()

# CSV upload path: open and send file

with open(file\_path, 'rb') as f:

files = {"file": (filename, f, "text/csv")}

data = {

"db\_type": db\_type,

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"process\_large\_files": process\_large\_files,

"use\_turbo": use\_turbo,

"batch\_size": batch\_size

}

response = requests.post(f"{API\_BASE\_URL}/run\_fast", files=files, data=data)

return response.json()

except Exception as e:

return {"error": f"API call failed: {str(e)}"}

# Deep Config Step-by-Step API Functions

def call\_deep\_config\_preprocess\_api(file\_path: str, filename: str, db\_config: dict = None):

"""Step 1: Preprocess data"""

try:

if db\_config and db\_config.get('use\_db'):

data = {

"db\_type": db\_config.get("db\_type"),

"host": db\_config.get("host"),

"port": db\_config.get("port"),

"username": db\_config.get("username"),

"password": db\_config.get("password"),

"database": db\_config.get("database"),

"table\_name": db\_config.get("table\_name")

}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/preprocess", data=data)

else:

with open(file\_path, 'rb') as f:

files = {"file": (filename, f, "text/csv")}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/preprocess", files=files)

return response.json()

except Exception as e:

return {"error": f"Preprocess API call failed: {str(e)}"}

def call\_deep\_config\_type\_convert\_api(type\_conversions: dict):

"""Step 2: Convert data types"""

try:

data = {"type\_conversions": json.dumps(type\_conversions)}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/type\_convert", data=data)

return response.json()

except Exception as e:

return {"error": f"Type convert API call failed: {str(e)}"}

def call\_deep\_config\_null\_handle\_api(null\_strategies: dict):

"""Step 3: Handle null values"""

try:

data = {"null\_strategies": json.dumps(null\_strategies)}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/null\_handle", data=data)

return response.json()

except Exception as e:

return {"error": f"Null handle API call failed: {str(e)}"}

def call\_deep\_config\_stopwords\_api(remove\_stopwords: bool):

"""Step 4: Remove stop words"""

try:

data = {"remove\_stopwords": remove\_stopwords}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/stopwords", data=data)

return response.json()

except Exception as e:

return {"error": f"Stopwords API call failed: {str(e)}"}

def call\_deep\_config\_normalize\_api(text\_processing: str):

"""Step 5: Text normalization"""

try:

data = {"text\_processing": text\_processing}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/normalize", data=data)

return response.json()

except Exception as e:

return {"error": f"Normalize API call failed: {str(e)}"}

def call\_deep\_config\_chunk\_api(chunk\_params: dict):

"""Step 6: Chunk data"""

try:

# Extract parameters from the dictionary

chunk\_method = chunk\_params.get("method", "fixed")

chunk\_size = chunk\_params.get("chunk\_size", 400)

overlap = chunk\_params.get("overlap", 50)

key\_column = chunk\_params.get("key\_column")

token\_limit = chunk\_params.get("token\_limit", 2000)

preserve\_headers = chunk\_params.get("preserve\_headers", True)

data = {

"chunk\_method": chunk\_method,

"chunk\_size": chunk\_size,

"overlap": overlap,

"token\_limit": token\_limit,

"preserve\_headers": preserve\_headers

}

if key\_column:

data["key\_column"] = key\_column

if chunk\_method == "semantic":

data["n\_clusters"] = chunk\_params.get("n\_clusters", 10)

response = requests.post(f"{API\_BASE\_URL}/deep\_config/chunk", data=data)

return response.json()

except Exception as e:

return {"error": f"Chunk API call failed: {str(e)}"}

def call\_deep\_config\_embed\_api(embed\_params: dict):

"""Step 7: Generate embeddings"""

try:

# Extract parameters from the dictionary

model\_name = embed\_params.get("model\_name", "paraphrase-MiniLM-L6-v2")

use\_openai = embed\_params.get("use\_openai", False)

openai\_api\_key = embed\_params.get("openai\_api\_key")

openai\_base\_url = embed\_params.get("openai\_base\_url")

batch\_size = embed\_params.get("batch\_size", 64)

use\_parallel = embed\_params.get("use\_parallel", True)

data = {

"model\_name": model\_name,

"use\_openai": use\_openai,

"batch\_size": batch\_size

}

if openai\_api\_key:

data["openai\_api\_key"] = openai\_api\_key

if openai\_base\_url:

data["openai\_base\_url"] = openai\_base\_url

response = requests.post(f"{API\_BASE\_URL}/deep\_config/embed", data=data)

return response.json()

except Exception as e:

return {"error": f"Embed API call failed: {str(e)}"}

def call\_deep\_config\_store\_api(store\_params: dict):

"""Step 8: Store embeddings"""

try:

data = {

"storage\_type": store\_params.get("storage\_type", "chroma"),

"collection\_name": store\_params.get("collection\_name", f"chunks\_{int(time.time())}"),

"persist\_directory": store\_params.get("persist\_directory", "./chroma\_db")

}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/store", data=data)

return response.json()

except Exception as e:

return {"error": f"Store API call failed: {str(e)}"}

# ---------- Main Application ----------

def main():

# Initialize session state

if 'current\_step' not in st.session\_state:

st.session\_state.current\_step = 0

if 'processing\_mode' not in st.session\_state:

st.session\_state.processing\_mode = None

if 'file\_uploaded' not in st.session\_state:

st.session\_state.file\_uploaded = False

if 'deep\_config\_data' not in st.session\_state:

st.session\_state.deep\_config\_data = {}

if 'fast\_mode\_results' not in st.session\_state:

st.session\_state.fast\_mode\_results = None

# Sidebar for mode selection

with st.sidebar:

st.markdown('<div class="section-header">Processing Mode</div>', unsafe\_allow\_html=True)

mode = st.radio(

"Choose Processing Mode:",

["Fast Mode", "Deep Config Mode"],

key="mode\_selector"

)

st.markdown("""

<div style="background: var(--ev-colors-primary); padding: 15px; border-radius: 6px; margin: 15px 0;">

<h4 style="color: var(--ev-colors-text); margin: 0 0 10px 0;">Mode Description</h4>

<p style="color: var(--ev-colors-tertiaryText); font-size: 0.9em; margin: 0;">

<strong>Fast Mode:</strong> Quick processing with default settings<br>

<strong>Deep Config:</strong> Step-by-step configuration with full control

</p>

</div>

""", unsafe\_allow\_html=True)

# Main content area

if mode == "Fast Mode":

render\_fast\_mode()

else:

render\_deep\_config\_mode()

def render\_fast\_mode():

"""Render the Fast Mode interface"""

st.markdown('<div class="section-header">🚀 Fast Processing Mode</div>', unsafe\_allow\_html=True)

# File upload section

st.markdown("""

<div class="custom-card">

<div class="card-title">📁 Data Source</div>

<div class="card-content">

Upload your CSV file or connect to a database for fast processing with optimized defaults.

</div>

</div>

""", unsafe\_allow\_html=True)

col1, col2 = st.columns(2)

with col1:

uploaded\_file = st.file\_uploader(

"Upload CSV File",

type=['csv'],

help="Upload CSV files up to 3GB in size"

)

with col2:

use\_db = st.checkbox("Use Database Connection")

db\_config = None

if use\_db:

db\_type = st.selectbox("Database Type", ["postgresql", "mysql", "sqlite"])

col1, col2 = st.columns(2)

with col1:

host = st.text\_input("Host", value="localhost")

port = st.number\_input("Port", value=5432 if db\_type == "postgresql" else 3306)

with col2:

username = st.text\_input("Username")

password = st.text\_input("Password", type="password")

database = st.text\_input("Database")

table\_name = st.text\_input("Table Name")

db\_config = {

"use\_db": True,

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

"table\_name": table\_name

}

# Processing options

st.markdown("""

<div class="custom-card">

<div class="card-title">⚙️ Processing Options</div>

<div class="card-content">

Configure how your data should be processed and embedded.

</div>

</div>

""", unsafe\_allow\_html=True)

col1, col2 = st.columns(2)

with col1:

use\_openai = st.checkbox("Use OpenAI Embeddings", help="Use OpenAI API for embeddings (more accurate but costs money)")

if use\_openai:

openai\_api\_key = st.text\_input("OpenAI API Key", type="password")

openai\_base\_url = st.text\_input("OpenAI Base URL (optional)", placeholder="https://api.openai.com/v1")

else:

openai\_api\_key = None

openai\_base\_url = None

process\_large\_files = st.checkbox("Process Large Files", value=True,

help="Enable for files larger than 100MB")

with col2:

use\_turbo = st.checkbox("Use Turbo Mode", value=True,

help="Faster processing with optimized settings")

batch\_size = st.number\_input("Batch Size", min\_value=32, max\_value=1024, value=256,

help="Larger batches = faster but more memory usage")

# Process button

if (uploaded\_file is not None or (use\_db and db\_config)) and st.session\_state.processing\_mode != "running":

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("🚀 Start Fast Processing", key="fast\_process", use\_container\_width=True):

st.session\_state.processing\_mode = "running"

st.session\_state.file\_uploaded = True

# Process the file

with st.spinner("Processing your data..."):

try:

if uploaded\_file:

# Save uploaded file temporarily

with tempfile.NamedTemporaryFile(delete=False, suffix='.csv') as tmp\_file:

tmp\_file.write(uploaded\_file.getvalue())

tmp\_path = tmp\_file.name

# Call FastAPI

result = call\_fast\_api(

tmp\_path,

uploaded\_file.name,

"csv",

None,

use\_openai,

openai\_api\_key,

openai\_base\_url,

process\_large\_files,

use\_turbo,

batch\_size

)

# Clean up temp file

os.unlink(tmp\_path)

else:

# Database processing

result = call\_fast\_api(

"", "", "", db\_config,

use\_openai, openai\_api\_key, openai\_base\_url,

process\_large\_files, use\_turbo, batch\_size

)

st.session\_state.fast\_mode\_results = result

st.session\_state.processing\_mode = "completed"

st.rerun()

except Exception as e:

st.error(f"Processing failed: {str(e)}")

st.session\_state.processing\_mode = None

st.markdown('</div>', unsafe\_allow\_html=True)

# Show results

if st.session\_state.processing\_mode == "completed" and st.session\_state.fast\_mode\_results:

results = st.session\_state.fast\_mode\_results

st.markdown("""

<div class="custom-card" style="border-left: 4px solid var(--ev-colors-success);">

<div class="card-title">✅ Processing Complete!</div>

</div>

""", unsafe\_allow\_html=True)

# Display results

col1, col2, col3 = st.columns(3)

with col1:

st.metric("Total Chunks", results.get('total\_chunks', 'N/A'))

with col2:

st.metric("Embedding Model", results.get('embedding\_model', 'N/A'))

with col3:

st.metric("Processing Time", f"{results.get('processing\_time', 0):.2f}s")

# Show sample chunks

if 'sample\_chunks' in results:

st.subheader("Sample Chunks")

for i, chunk in enumerate(results['sample\_chunks'][:3]):

with st.expander(f"Chunk {i+1}"):

st.text(chunk)

def render\_deep\_config\_mode():

"""Render the Deep Config Mode interface"""

st.markdown('<div class="section-header">🔧 Deep Configuration Mode</div>', unsafe\_allow\_html=True)

st.markdown("""

<div class="custom-card">

<div class="card-title">Step-by-Step Configuration</div>

<div class="card-content">

Configure each aspect of your data processing pipeline with full control over every parameter.

Follow the steps below to optimize your chunking and embedding process.

</div>

</div>

""", unsafe\_allow\_html=True)

# Step navigation

steps = [

"1. Data Source & Preprocessing",

"2. Data Type Conversion",

"3. Null Value Handling",

"4. Stop Words Removal",

"5. Text Normalization",

"6. Chunking Strategy",

"7. Embedding Generation",

"8. Storage & Export"

]

# Create step indicators

cols = st.columns(len(steps))

for i, (col, step) in enumerate(zip(cols, steps)):

with col:

status = "completed" if i < st.session\_state.current\_step else "pending"

if i == st.session\_state.current\_step:

status = "running"

st.markdown(f"""

<div class="process-step {status}">

<div style="font-size: 0.8em; font-weight: 600; color: var(--ev-colors-text);">{step}</div>

</div>

""", unsafe\_allow\_html=True)

# Render current step

if st.session\_state.current\_step == 0:

render\_step\_0()

elif st.session\_state.current\_step == 1:

render\_step\_1()

elif st.session\_state.current\_step == 2:

render\_step\_2()

elif st.session\_state.current\_step == 3:

render\_step\_3()

elif st.session\_state.current\_step == 4:

render\_step\_4()

elif st.session\_state.current\_step == 5:

render\_step\_5()

elif st.session\_state.current\_step == 6:

render\_step\_6()

elif st.session\_state.current\_step == 7:

render\_step\_7()

def render\_step\_0():

"""Step 0: Data Source & Preprocessing"""

st.markdown("""

<div class="custom-card">

<div class="card-title">📊 Data Source Selection</div>

<div class="card-content">

Choose your data source and configure basic preprocessing options.

</div>

</div>

""", unsafe\_allow\_html=True)

col1, col2 = st.columns(2)

with col1:

uploaded\_file = st.file\_uploader(

"Upload CSV File",

type=['csv'],

key="deep\_config\_upload",

help="Upload your dataset for processing"

)

with col2:

use\_db = st.checkbox("Use Database Connection", key="deep\_config\_db")

db\_config = None

if use\_db:

db\_type = st.selectbox("Database Type", ["postgresql", "mysql", "sqlite"], key="db\_type")

col1, col2 = st.columns(2)

with col1:

host = st.text\_input("Host", value="localhost", key="db\_host")

port = st.number\_input("Port", value=5432 if db\_type == "postgresql" else 3306, key="db\_port")

with col2:

username = st.text\_input("Username", key="db\_user")

password = st.text\_input("Password", type="password", key="db\_pass")

database = st.text\_input("Database", key="db\_name")

table\_name = st.text\_input("Table Name", key="db\_table")

db\_config = {

"use\_db": True,

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

"table\_name": table\_name

}

if (uploaded\_file or (use\_db and db\_config)):

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Process Data", key="step0\_process", use\_container\_width=True):

with st.spinner("Preprocessing data..."):

try:

if uploaded\_file:

# Save uploaded file temporarily

with tempfile.NamedTemporaryFile(delete=False, suffix='.csv') as tmp\_file:

tmp\_file.write(uploaded\_file.getvalue())

tmp\_path = tmp\_file.name

result = call\_deep\_config\_preprocess\_api(tmp\_path, uploaded\_file.name)

os.unlink(tmp\_path)

else:

result = call\_deep\_config\_preprocess\_api("", "", db\_config)

if "error" not in result:

st.session\_state.deep\_config\_data['preprocess'] = result

st.session\_state.current\_step = 1

st.rerun()

else:

st.error(f"Preprocessing failed: {result['error']}")

except Exception as e:

st.error(f"Error during preprocessing: {str(e)}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_1():

"""Step 1: Data Type Conversion"""

st.markdown("""

<div class="custom-card">

<div class="card-title">🔄 Data Type Conversion</div>

<div class="card-content">

Configure how each column should be treated during processing.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'preprocess' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 0 first.")

return

preprocess\_data = st.session\_state.deep\_config\_data['preprocess']

columns = preprocess\_data.get('columns', [])

sample\_data = preprocess\_data.get('sample\_data', {})

st.subheader("Column Type Configuration")

type\_conversions = {}

for col in columns:

col1, col2 = st.columns([2, 1])

with col1:

st.text(f"Column: {col}")

if col in sample\_data:

st.caption(f"Sample: {sample\_data[col]}")

with col2:

col\_type = st.selectbox(

f"Type for {col}",

["text", "numeric", "categorical", "datetime", "ignore"],

key=f"type\_{col}"

)

type\_conversions[col] = col\_type

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Apply Type Conversions", key="step1\_apply", use\_container\_width=True):

with st.spinner("Applying type conversions..."):

result = call\_deep\_config\_type\_convert\_api(type\_conversions)

if "error" not in result:

st.session\_state.deep\_config\_data['type\_convert'] = result

st.session\_state.current\_step = 2

st.rerun()

else:

st.error(f"Type conversion failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_2():

"""Step 2: Null Value Handling"""

st.markdown("""

<div class="custom-card">

<div class="card-title">🔄 Null Value Handling</div>

<div class="card-content">

Configure how to handle missing values in your dataset.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'type\_convert' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 1 first.")

return

type\_data = st.session\_state.deep\_config\_data['type\_convert']

columns = type\_data.get('columns', [])

null\_counts = type\_data.get('null\_counts', {})

st.subheader("Null Value Strategies")

null\_strategies = {}

for col in columns:

null\_count = null\_counts.get(col, 0)

col1, col2, col3 = st.columns([2, 1, 2])

with col1:

st.text(f"Column: {col}")

st.caption(f"Null values: {null\_count}")

with col2:

if null\_count > 0:

strategy = st.selectbox(

f"Strategy for {col}",

["fill\_with\_mean", "fill\_with\_median", "fill\_with\_mode", "fill\_with\_value", "drop\_rows"],

key=f"null\_{col}"

)

else:

strategy = "none"

st.info("No null values")

with col3:

if strategy == "fill\_with\_value":

fill\_value = st.text\_input(f"Fill value for {col}", key=f"fill\_{col}")

null\_strategies[col] = {"strategy": strategy, "fill\_value": fill\_value}

else:

null\_strategies[col] = {"strategy": strategy}

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Apply Null Handling", key="step2\_apply", use\_container\_width=True):

with st.spinner("Applying null handling strategies..."):

result = call\_deep\_config\_null\_handle\_api(null\_strategies)

if "error" not in result:

st.session\_state.deep\_config\_data['null\_handle'] = result

st.session\_state.current\_step = 3

st.rerun()

else:

st.error(f"Null handling failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_3():

"""Step 3: Stop Words Removal"""

st.markdown("""

<div class="custom-card">

<div class="card-title">🗑️ Stop Words Removal</div>

<div class="card-content">

Configure stop words removal from text columns.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'null\_handle' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 2 first.")

return

remove\_stopwords = st.checkbox(

"Remove Stop Words from Text Columns",

value=True,

help="Remove common stop words to improve embedding quality"

)

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Apply Stop Words Removal", key="step3\_apply", use\_container\_width=True):

with st.spinner("Processing stop words..."):

result = call\_deep\_config\_stopwords\_api(remove\_stopwords)

if "error" not in result:

st.session\_state.deep\_config\_data['stopwords'] = result

st.session\_state.current\_step = 4

st.rerun()

else:

st.error(f"Stop words removal failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_4():

"""Step 4: Text Normalization"""

st.markdown("""

<div class="custom-card">

<div class="card-title">📝 Text Normalization</div>

<div class="card-content">

Configure text normalization options for better embedding quality.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'stopwords' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 3 first.")

return

text\_processing = st.selectbox(

"Text Processing Level",

[

"minimal",

"standard",

"aggressive"

],

help="Minimal: lowercase only | Standard: lowercase + punctuation removal | Aggressive: full normalization"

)

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Apply Text Normalization", key="step4\_apply", use\_container\_width=True):

with st.spinner("Applying text normalization..."):

result = call\_deep\_config\_normalize\_api(text\_processing)

if "error" not in result:

st.session\_state.deep\_config\_data['normalize'] = result

st.session\_state.current\_step = 5

st.rerun()

else:

st.error(f"Text normalization failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_5():

"""Step 5: Chunking Strategy"""

st.markdown("""

<div class="custom-card">

<div class="card-title">✂️ Chunking Strategy</div>

<div class="card-content">

Configure how your data should be split into chunks for embedding.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'normalize' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 4 first.")

return

col1, col2 = st.columns(2)

with col1:

chunk\_method = st.selectbox(

"Chunking Method",

["fixed", "semantic", "recursive"],

help="Fixed: equal-sized chunks | Semantic: content-based grouping | Recursive: hierarchical splitting"

)

chunk\_size = st.number\_input(

"Chunk Size (characters)",

min\_value=100,

max\_value=2000,

value=400,

help="Target size for each chunk"

)

overlap = st.number\_input(

"Overlap Size",

min\_value=0,

max\_value=200,

value=50,

help="Overlap between consecutive chunks"

)

with col2:

token\_limit = st.number\_input(

"Token Limit",

min\_value=100,

max\_value=4000,

value=2000,

help="Maximum tokens per chunk (for token-based models)"

)

preserve\_headers = st.checkbox(

"Preserve Headers in Chunks",

value=True,

help="Include column headers in each chunk for context"

)

if chunk\_method == "semantic":

n\_clusters = st.number\_input(

"Number of Clusters",

min\_value=2,

max\_value=50,

value=10,

help="Number of semantic clusters to create"

)

chunk\_params = {

"method": chunk\_method,

"chunk\_size": chunk\_size,

"overlap": overlap,

"token\_limit": token\_limit,

"preserve\_headers": preserve\_headers

}

if chunk\_method == "semantic":

chunk\_params["n\_clusters"] = n\_clusters

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Apply Chunking", key="step5\_apply", use\_container\_width=True):

with st.spinner("Chunking data..."):

result = call\_deep\_config\_chunk\_api(chunk\_params)

if "error" not in result:

st.session\_state.deep\_config\_data['chunk'] = result

st.session\_state.current\_step = 6

st.rerun()

else:

st.error(f"Chunking failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_6():

"""Step 6: Embedding Generation"""

st.markdown("""

<div class="custom-card">

<div class="card-title">🔤 Embedding Generation</div>

<div class="card-content">

Configure embedding model and parameters for vector generation.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'chunk' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 5 first.")

return

col1, col2 = st.columns(2)

with col1:

model\_name = st.selectbox(

"Embedding Model",

[

"paraphrase-MiniLM-L6-v2",

"all-MiniLM-L6-v2",

"paraphrase-multilingual-MiniLM-L12-v2",

"custom"

],

help="Choose the embedding model for vector generation"

)

use\_openai = st.checkbox(

"Use OpenAI Embeddings",

help="Use OpenAI API for embeddings (requires API key)"

)

if use\_openai:

openai\_api\_key = st.text\_input("OpenAI API Key", type="password")

openai\_base\_url = st.text\_input("OpenAI Base URL (optional)", placeholder="https://api.openai.com/v1")

else:

openai\_api\_key = None

openai\_base\_url = None

with col2:

batch\_size = st.number\_input(

"Batch Size",

min\_value=8,

max\_value=256,

value=64,

help="Number of chunks to process in each batch"

)

use\_parallel = st.checkbox(

"Use Parallel Processing",

value=True,

help="Process multiple batches in parallel for speed"

)

embed\_params = {

"model\_name": model\_name,

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"batch\_size": batch\_size,

"use\_parallel": use\_parallel

}

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Generate Embeddings", key="step6\_apply", use\_container\_width=True):

with st.spinner("Generating embeddings..."):

result = call\_deep\_config\_embed\_api(embed\_params)

if "error" not in result:

st.session\_state.deep\_config\_data['embed'] = result

st.session\_state.current\_step = 7

st.rerun()

else:

st.error(f"Embedding generation failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

def render\_step\_7():

"""Step 7: Storage & Export"""

st.markdown("""

<div class="custom-card">

<div class="card-title">💾 Storage & Export</div>

<div class="card-content">

Configure how and where to store your processed chunks and embeddings.

</div>

</div>

""", unsafe\_allow\_html=True)

if 'embed' not in st.session\_state.deep\_config\_data:

st.warning("Please complete Step 6 first.")

return

col1, col2 = st.columns(2)

with col1:

storage\_type = st.selectbox(

"Storage Type",

["chroma", "json", "csv", "parquet"],

help="Choose where to store the processed data"

)

collection\_name = st.text\_input(

"Collection Name",

value=f"chunks\_{int(time.time())}",

help="Name for the stored collection"

)

with col2:

persist\_directory = st.text\_input(

"Storage Directory",

value="./chroma\_db",

help="Directory to store the processed data"

)

store\_params = {

"storage\_type": storage\_type,

"collection\_name": collection\_name,

"persist\_directory": persist\_directory

}

col1, col2 = st.columns(2)

with col1:

st.markdown('<div class="primary-button">', unsafe\_allow\_html=True)

if st.button("Store Results", key="step7\_store", use\_container\_width=True):

with st.spinner("Storing results..."):

result = call\_deep\_config\_store\_api(store\_params)

if "error" not in result:

st.session\_state.deep\_config\_data['store'] = result

st.success("✅ Processing completed successfully!")

# Show summary

st.subheader("Processing Summary")

summary\_data = st.session\_state.deep\_config\_data

col1, col2, col3 = st.columns(3)

with col1:

st.metric("Total Chunks", summary\_data.get('chunk', {}).get('total\_chunks', 'N/A'))

with col2:

st.metric("Embedding Model", summary\_data.get('embed', {}).get('model\_used', 'N/A'))

with col3:

st.metric("Storage", storage\_type)

else:

st.error(f"Storage failed: {result['error']}")

st.markdown('</div>', unsafe\_allow\_html=True)

with col2:

if st.button("Start Over", key="step7\_restart", use\_container\_width=True):

st.session\_state.current\_step = 0

st.session\_state.deep\_config\_data = {}

st.rerun()

if \_\_name\_\_ == "\_\_main\_\_":

main()